Generalizing the Herding Attack to Concatenated Hashing Schemes*

Orr Dunkelman[1] Bart Preneel[1]

[1] Katholieke Universiteit Leuven
Department of Electrical Engineering ESAT/SCD-COSIC
Kasteelpark Arenberg 10, B-3001 Leuven-Heverlee, Belgium
{orr.dunkelman,bart.preneel}@esat.kuleuven.be

Abstract. In this paper we extend the herding attacks for concatenated hash functions, i.e., hash functions of the form $h(x) = h_1(x)||h_2(x)$. Our results actually apply a much larger set of hash functions. We show that even when the compression function of $h_(\cdot)$ can be written as two (or more) data paths, where one data path is not affected by the second (while the second may depend on the first), then the generalized herding attack can be applied. This result along with Joux's original observations show that schemes that aim to improve the resistance of hash functions against these attacks, must use diffusion between the various data paths.

Keywords: Concatenated hash functions, Multi-Collisions, Herding attack.

1 Introduction

Most compression functions used in real life applications are iterated, i.e., iterate a compression function that accepts a message blocks of a fixed size. The Merkle-Damgård construction is the most widely used transformation of cryptographic secure compression functions into cryptographic hash functions [5, 13]. The construction suggests a simple transformation that maintains the collision resistance property of the underlying compression function. For years it was widely believed that the transformation maintains also the pre-image resistance of the underlying compression function as well as the second pre-image resistance.

However, in recent years, several counter examples for these beliefs were suggested. The first evidence for this was the works of Dean [6]. Dean showed that if fix-points of the compression function can be easily found, then second pre-image attacks on Merkle-Damgård hash functions can be mounted using $O(n \cdot 2^{n/2})$ time and $O(n \cdot 2^{n/2})$ memory (where n is the chaining value and the digest size). The attack uses expandable messages, i.e., messages that can be

* This work was supported in part by the Concerted Research Action (GOA) Ambiorics 2005/11 of the Flemish Government and by the IAP Programme P6/26 BCRYPT of the Belgian State (Belgian Science Policy).

expanded without changing their chaining value. A downside for Dean's attack is the fact that the original message and the second pre-image are both relatively long messages (of length $O(2^{n/2})$ blocks).

Another recent result on iterated hashing is the seminal work of Joux on finding multi-collisions [9]. The method proposed by Joux is based on finding a collision for each of the t blocks separately. Then, by combining any of the possible colliding blocks, one can find 2^t messages that are hashed to the same value.[1]

Later, Kelsey and Schneier have shows that second pre-image attacks on long messages can be mounted even when there are no easy to find fix-points [11]. Their improvement is based on replacing the need for fix-points with a generation of an expandable message using Joux's multi-collision attack.

These second pre-image attacks work in the environment of relatively long messages. In [10], Kelsey and Kohno showed that using a simple pre-computation, it is possible to reduce the time requirements of pre-image attacks (in some sense) of relatively short messages, while keeping the time complexity of the attack much below $O(2^n)$. The herding attack is a time-memory computational trick that allows the attacker to commit to a digest value, and then produce a message with the corresponding digest, even when the message has a prefix (of limited length) dictated to the attacker by an outside source.

Besides finding multi-collision, Joux [9] has also analyzed the case of two (or more) concatenated hashing, i.e., hash functions $h(x)$ that can be written as $h(x) = h_1(x)||h_2(x)$. Joux showed that if $h_1(\cdot)$ and $h_2(\cdot)$ are iterative[2] then the time complexity of finding collisions in $h(\cdot)$ is very close to the time complexity of finding collisions in $h_1(\cdot)$ or $h_2(\cdot)$ (up to a linear factor which depends on the lengths of the outputs of $h_1(\cdot)$ and $h_2(\cdot)$). The same holds for the problem of finding pre-images for this kind of hash functions.

In this work we extend the herding attacks to hash functions where the compression function can be written as two (or more) data paths, such that one data path is not affected by others, (but affects others). This follow the fact that Joux's multi-collision attack still succeeds for such functions. Along with Joux's prior results this is an important observation for hash function designers who try to increase the security of hash functions without changing the compression function. This can be done by adding data paths which interact between themselves (i.e., have several compression function calls for each message block). Such a family of constructions is presented in [7] under the name 3C. While the paper presenting the 3C family states that Joux's multi-collision attack is easily scaled to the 3C family, it claims that attacks which are built over this attack are not expected to work.

This paper is organized as follows: In Section 2 we describe Joux's multi-collision attack and the attacks on concatenated hash functions along with the 2nd pre-image attack by Kelsey and Schneier and the Herding attack by Kelsey

[1] We note that in [4] the same basic idea was used in a pre-image attack against a special kind of hash function.

[2] We note that even if only one of the hash functions is iterative Joux's results hold.

and Kohno. In Section 3 we give the explicit algorithms of Joux's for concatenation done inside the compression function from one data path to the others. We follow to show the generalization of the herding attack to the same scenario in Section 4. Finally, Section 5 summarizes the paper.

2 Previous Work

The Merkle-Damgård construction is the most common way to transform a compression function $C : \{0,1\}^n \times \{0,1\}^m \rightarrow \{0,1\}^n$ into a hash function $h_C(\cdot)$ [5, 13]. Throughout this paper n denotes the size of the chaining value, and m denotes the block size for the compression function.

The message M is padded with its length (after additional padding to make the message a multiple of the block size m after the final padding), and the message is divided into blocks of m bits each, $M = M_1 M_2 M_3 \dots M_t$. An initial chaining value $h_0 = IV \in \{0,1\}^n$ is set for the hash function (also called initialization vector) and the following process is repeated t times:

$$h_i = C(h_{i-1}, M_i)$$

The final h_t is outputed as the hash value.

2.1 Joux's Multi-Collisions Attack on Iterated Hash Functions

In [9] it was observed that finding multiple collisions in an iterated hash function is not much harder than finding one collision. The key observation behind the attack is the fact that in iterative hashing schemes, such as the Merkle-Damgård, it is possible to find a collision for each block, e.g., for any h_{i-1} finding M_i and M_i^* such that $C(h_{i-1}, M_i) = C(h_{i-1}, M_i^*)$. Finding t such one block collisions (each starting from the chaining value produced by the previous block collision) it is possible to construct 2^t messages with the same hash value by selecting for the ith block of the message either M_i or M_i^*.

2.2 Joux's Collision Attack on Concatenated Hash Functions

Joux's multi-collision attack can be used to attack concatenated hash functions, i.e., $h(x) = h_1(x)||h_2(x)$. Such hash functions are as secure against collision attacks and pre-image attacks as the stronger of the two underlying hash functions.[3] Let n_1 and n_2 be the chaining value lengths of $h_1(\cdot)$ and $h_2(\cdot)$, respectively.

To produce the collision in $h(\cdot)$, the attacker finds $2^{n_2/2}$ collisions in $h_1(\cdot)$ using the multi-collision attack in time complexity of $O(n_2/2 \cdot 2^{n_1/2})$ calls to $h_1(\cdot)$. These colliding messages produce $2^{n_2/2}$ messages with the same output of $h_1(\cdot)$, and by the birthday paradox, we expect also one collision in the value of

[3] We note that the Joux showed that the result is extendible to concatenation of several iterative hash functions as well.

$h_2(\cdot)$. This collision is also a collision of $h(\cdot)$ itself which was found using about $O(n_2/2 \cdot 2^{n_1/2})$ calls to the compression function of $h_1(\cdot)$ and $O(n_2/2 \cdot 2^{n_2/2})$ calls to the compression function of $h_2(\cdot)$ (after taking into consideration that the resulting values are of $n_2/2$ blocks each).

It is easy to see that for each additional compression function which is concatenated, we need to produce $2^{n_i/2}$ collisions for the previous hash functions. This can be done efficiently by starting from the already found collision, and transforming it into a multi-collision in a similar manner. The time complexity of the attack in the case of k hash functions is

$$\left(\prod_{i=2}^{k} n_k\right) \cdot \left(\sum_{i=1}^{k} 2^{n_k/2}\right)$$

2.3 Joux's Pre-image Attack on Concatenated Hash Functions

Another interesting result found in [9] is an algorithm for finding pre-images for concatenated hash functions. For the case of two hash functions there is a special algorithm, but we shall give the more general algorithm here.

Let IV_i be the initialization vector of $h_i(\cdot)$. The attacker starts by finding two message blocks s_1^0 and s_1^1 such that $IV_1 = h(IV_1, s_1^0) = h(IV_1, s_1^1)$.[4]

Now examine $h_2(\cdot)$. The attacker then looks for two sequences of message blocks s_2^0, s_2^1 (with the same length) of message blocks taken from $\{s_1^0, s_1^1\}$ each time such that $IV_2 = h_2(IV_2, s_2^0)$ and $IV_2 = h_2(IV_2, s_2^1)$. We note that both sequences satisfy that $IV_1 = h_1(IV_1, s_2^0)$ and $IV_1 = h_1(IV_1, s_2^1)$, due to the way they were constructed. Each new hash function $h_i(\cdot)$ requires finding two sequences based on the two sequences found for $h_{i-1}(\cdot)$ such that $h_i(\cdot)$ produces a fix-point.

Now given $h(x) = y = (y_1, y_2, \ldots, h_k)$, the attacker searches for a message block B for which $h_1(IV_1, B) = y_1$. Once such a value B is found, the attacker searches for a sequence s of the blocks $\{s_1^0, s_1^1\}$ such that $h_2(IV_2, \{s_1^0, s_1^1\}^+ || B) = y_2$, and continue in that manner, until she knows how to reach from IV_i to the corresponding y_i for all i's. Then, the attacker just fixes the lengths of the messages (using the fact that the sequences produce fix-points). We note that there is no need to find fix-points for $h_k(\cdot)$.

There is an important technical aspect following the Merkle-Damgård strengthening. The block B should contain the encoding of the message length, and the length of the message is not necessarily known in advance. However, the length of B can be estimated in advance (due to the number of concatenations expected for each hash function). Moreover, in the worst case, a different block B can be chosen if the need arises.

The time complexity of the attack for k hash functions is $2 \cdot 2^{n_1} + 2 \cdot n_2 \cdot 2^{n_2} + \ldots + 2\prod_{i=2}^{k-1} n_i \cdot 2^{n_{k-1}}$ for finding the fix-points, and $2^{n_1} + (n_2 + 1) \cdot 2^{n_2} +$

[4] We note that such an s_1^0 and s_1^1 does not necessarily exist, but in that case, one can treat them as made of two (or more) blocks. Another possibility is to fix a first block $\hat{M}$ and concatenate it before all messages, thus, changing the value of IV_1 into a different value for which such pre-images may exist.

$\ldots + (1 + \prod_{i=2}^{k} n_i) \cdot 2^{n_k}$ for the final phase of identifying B and the number of times each fix-point is to be used.

2.4 The Long 2nd Pre-image Attack

It was widely believed by the cryptographic community that the security proof of the Merkle-Damgård construction applies also to second pre-image attacks. However, Dean [6] noticed that this is not true for long messages if the compression function has easy to find fix-points. His observations were later generalized by Kelsey and Schneier that used the multi-collision technique to replace the need of easily found fix-points [11].

A simplified variant of the attack is as follows: consider a long message $M = M_1 M_2 \ldots M_l$ that is processed using $h(\cdot)$, a Merkle-Damgård hash function, when the message length is not padded (i.e., without the Merkle-Damgård strengthening). An attacker that wishes to construct a message M^* such that $h(M) = h(M^*)$ can randomly select messages M' until $h(M')$ equals to any of the l chaining values found during the computation of $h(M)$. Once such an instance is found, the attacker can concatenate to M' the message blocks of M that are hashed starting from the given chaining value, resulting in M^* s.t. $h(M) = h(M^*)$. This attack is foiled by the Merkle-Damgård strengthening, as the message length is padded, changing the last block that enters the compression function.

Dean proposed to use to use fix-points to overcome this problem [6]. Assume that the compression function $C(\cdot)$ is such that finding fix-points is easy, i.e., it is easy to find many (h, M) pairs satisfying $h = C(h, M)$. This is the case for the Davies-Meyer construction that takes a block cipher E that accept n-bit plaintexts (chaining values) and m-bit keys (message blocks) and sets

$$h_i = C(h_{i-1}, M) = E_M(h_{i-1}) \oplus h_{i-1}.$$

For such a compression function it is easy to find fix-points by computing $h = E_M^{-1}(0)$ for randomly selected messages M.

Given these easily found fix-points Dean's attack is as follows:

1. Find a set $O(2^{n/2})$ fix-points denoted by $A = \{(h, m)\}$.
2. Select a set of $O(2^{n/2})$ one block messages, and compute their chaining value denoted by $B = \{C(IV, \tilde{m})\}$.
3. Once a collision between a chaining value and a fix-point, i.e., between chaining values in A and in B, is found, the previous attack is repeated (i.e., trying to add blocks that cause the same chaining values as the original message).

Once such a message is found, it is easy to expand the number of blocks in the message to the appropriate length by repeating the fix-point as many times as needed. The time complexity of Dean's attack is $O(2^n/2)$ calls for the compression function, $O(2^n/2)$ queries for the fix-points oracle, and $O(2^{n-r}$ compression function calls to find a second pre-image for a $2^r + 1$ block message.

Kelsey and Schneier transformed the attack to the case were fix-points are not easily found. While Dean's expandable message could be extended by a concatenation of one message block, in their attack they use the multi-collision technique to produce an expandable message. They replace the first two steps in Dean's attack in the following procedure. In each iteration $1 \leq i \leq t$ a collision between a one block message and a $2^{i-1}+1$ block message is found. This procedure finds a chaining value that can be reached by messages of lengths between t and 2^t+t-1 blocks. Then, from this chaining value the third step of Dean's attack is executed, and the length of the found message is controlled by the expandable prefix.

The time complexity of the long messages pre-image attack is $(2^t+2t+1)\cdot 2^{n/2}$ compression function calls for generating the expandable message, and then 2^{n-r} to find a colliding chaining value with a message of length 2^r+t.

2.5 The Herding Attack

Kelsey and Kohno have observed that it is possible to perform a time-memory tradeoff for several instances of pre-image attacks [10]. In their attack, an attacker commits to a public digest value that corresponds to some meaningful message, e.g., a prediction of the outcome of NIST's hash function competition. After the announcement of the result, the attacker publishes a message that has the pre-published digest value and contains the correct information along with some suffix.

The attack is based on selecting the digest value carefully, helping the attacker to perform a pre-image attack on this value. The main idea behind this attack is to store $T = 2^t$ possible chaining values h_i for which the attacker knows how to reach the published digest value h_f. From each such chaining value, $O(2^{n/2-t/2+1/2})$ one block message are hashed, resulting in collisions between pairs of chaining values and messages (h_i, m_j). The attacker continues iteratively to find collisions between the new chaining values, until the attacker finds the final collision. These values (and the corresponding messages) are called a diamond structure, as presented in Figure 2.5.

The attacker then publishes the last chaining value as a target. When trying to find a pre-image to this target, the attacker needs to perform only $O(2^{n-t})$ operations until finding a message that has a chaining value among the 2^t original values.

We note that unlike the previous attacks that require long messages, this attack appends relatively short suffix (about t blocks) to the "real" message. We also note, that the total time complexity of the attack is about $O(2^{n/2+t/2})$ off-line operations for the first step of the attack (which is the most time consuming[5]) and $O(2^{n-t})$ online operations for the second step. For $t = n/3$ the overall time complexity of this attack is $O(2^{2n/3})$ for finding a pre-image.

[5] Each new step requires about $\sqrt{2}$ times more messages and has twice as little chaining values.

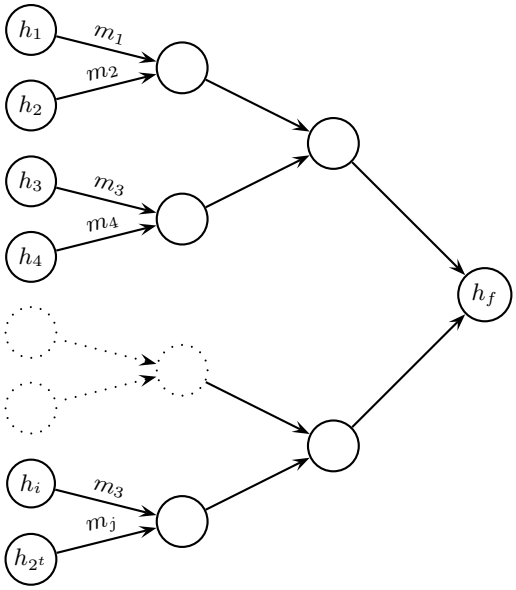

Fig. 1. A Diamond Structure

We note that [10] contains also an elongated version that uses long messages. In this version, from each starting chaining value, a message of 2^r blocks is compressed. Thus, the linking message has 2^{t+r} possible chaining values as targets. In exchange, to compensate for the unknown length, in the online phase of the attack, the attacker first generates an expandable message, and only then tries to generate a linking message. This approach increases the pre-computation phase's complexity by a factor of 2^{t+r} more compression function calls, in exchange of reducing the online phase to $O(2^{n-t-r})$ operations.

3 Extending Joux's Attacks on Concatenated Hashing Schemes

3.1 Motivation

Due to the recent attacks on specific hash functions, as well as the generic attacks on the iterated Merkle-Damgård construction, various fortifications to Merkle-Damgård were suggested. Some suggestions deal with pre-processing of the message, but as shown in [8], as long as the each bit is used a finite number of times, Joux's multi-collision attack still works.

A different approach was taken in [7]. In this approach, in order to add protection against the specific attacks on hash functions (specifically collision attacks that use more than one block as those in [1, 2, 15–18]), the chaining

value is composed of several values. Several variants were suggested, including:

$$3C((cv_1, cv_2), M_i) = (C(cv_1, M_i), cv_2 \oplus C(cv_1, M_i)),$$
$$3CG((cv_1, cv_2), M_i) = (C(cv_1, M_i), f(cv_2, C(cv_1, M_i))),$$

where C is the underlying compression function, cv_i are the chaining values and f is some function. That way the time of the compression function is not greatly increased, and the current multi-block attacks are harder to implement.

As noted in [7], this suggestion is not secure against Joux's multi-collision attack, but it's expected to be more secure against the herding and the long message second pre-image attacks. The authors claim that this is under the assumption that the attacker wishes to attack the entire construction simultaneously, while it is obvious that an attacker might attack each chaining value independently (as noted in a comment due to John Kelsey).

3.2 The Settings

Following the 3C family, we have decided to investigate iterated hash functions $h(\cdot)$ such that their compression function can be described as:

$$C((cv_1, cv_2, \ldots, cv_k), M_i) = (c_1(cv_1, M_i), c_2(cv_1, cv_2, M_i), c_3(cv_1, cv_2, cv_3, M_i)\ldots),$$

where c_i are compression functions. We note that the fact that $c_2(\cdot)$ accepts cv_1 and M_i, allows it to use any intermediate value (or final value) produced during the computation of $c_1(cv_1, M_i)$. This scenario corresponds to several data paths which are ordered such that each data path affects only data paths with a larger index.

We shall assume without loss of generality that $c_i(\cdot)$ all have the same block size (for sake of simplicity), as otherwise, we can discuss a block size which is the least common multiplier of all block sizes. We also denote the length of the chaining value cv_i by n_i. We shall assume that n_i is even (the results for odd values of n_i can be easily derived from the even case).

We shall start by giving the exact algorithms for Joux's attacks in this case, as the multi-collision is the basic step used in our generalized herding attack.

3.3 Extending Joux's Multi-Collision Attack

The extended algorithm is very similar to the original algorithm proposed by Joux in [9]. The only difference is during the computations related to $c_i(\cdot)$ for $i \geq 2$ where the information from the previous functions is included.

The following algorithm deals with the case of $C((cv_1, cv_2), M_i) = (c_1(cv_1, M_i), c_2(cv_1, cv_2, M_i))$ with initial values (IV_1, IV_2), and can be easily extended to any number of compression functions:

1. Denote $cv_1^0 = IV_1$.
2. For $i = 1, \ldots, n_2/2$:

(a) Find two message blocks M_i^0 and M_i^1 such that $c_1(cv_i, M_i^0) = c_1(cv_i, M_i^1)$, and set $cv_{i+1} = c_1(cv_i, M_i^0)$.

3. For all $2^{n_2/2}$ messages of the form $M^{e_1 e_2 \ldots e_{n_2/2}} \stackrel{def}{=} M_1^{e_1} || M_2^{e_2} || \ldots || M_{e_{n_2/2}/2}^{e_{n_2/2}}$ for $e_i \in \{0, 1\}$, compute $c_2(cv_1, cv_2, M^{e_1 e_2 \ldots e_{n_2/2}})$.
4. By the birthday paradox we expect a collision in the computed hash values of $c_2(\cdot)$ while we are assured to have a collision in the output of $c_1(\cdot)$.

The time complexity of the attack is $n_2/2 \cdot 2^{n_1/2}$ applications of $c_1(\cdot)$ on one message blocks, and an additional $2^{n_2/2}$ applications of $c_2(\cdot)$ on messages of $n_2/2$ blocks. The total time complexity is $n_2/2 \cdot (2^{n_1/2} + 2^{n_2/2})$ compression function calls, which is only linearly $n_2/2$ times worse than the number of calls required to generate a collision in $c_1(\cdot)$ or in $c_2(\cdot)$.

The time complexity of this attack for k compression functions with respective output lengths of $n_1, n_2, \ldots n_k$ is:

$$\left[\prod_{i=2}^{k} \left(\frac{n_i}{2}\right)\right] \cdot \left[\sum_{i=1}^{k} \left(2^{n_i/2}\right)\right].$$

This work factor as the same as for regular concatenated hash functions.

3.4 Extending Joux's Pre-image Attack

As before, Joux's pre-image attack can be easily adapted to the family of hash functions that we study. The following algorithm finds pre-images in hash functions of the form of $C((cv_1, cv_2), M_i) = (c_1(cv_1, M_i), c_2(cv_1, cv_2, M_i))$ with initial values (IV_1, IV_2), and can be easily extended to any number of compression functions. Let (y_1, y_2) be the required hash value, apply the following algorithm:

1. Find two messages M_0 and M_1 such that $c_1(IV_1, M_0) = c_1(IV_1, M_1) = IV_1$.
2. Find B, a message block which encode a message length of $n_2 + 1$ blocks, such that $c_1(IV_1, B) = y_1$.
3. For each of the 2^{n_2} possible messages $M^{e_1, e_2, \ldots, e_{n_2}} = M_{e_1} || M_{e_2} || \ldots || M_{e_{n_2}} || B$ for $e_i \in \{0, 1\}$, compute $c_2(IV_1, IV_2, M^{e_1, e_2, \ldots, e_{n_2}})$. Output $M^{e_1, e_2, \ldots, e_{n_2}}$ for which $c_2(IV_1, IV_2, M^{e_1, e_2, \ldots, e_{n_2}}) = y_2$.

The time complexity of the above algorithm is expected to be $2 \cdot 2^{n_1}$ calls to $c_1(\cdot)$ for finding M_0 and M_1. An additional 2^{n_1} calls for $c_1(\cdot)$ are expected for finding B. Finally, 2^{n_2} calls for $c_2(\cdot)$ with blocks of length $n_2 + 1$ are expected for the final step of the attack. Thus, the total number of compression functions calls is:

$$2 \cdot 2^{n_1} + 2^{n_1} + (n_2 + 1) \cdot 2^{n_2} = 3 \cdot 2^{n_1} + (n_2 + 1) \cdot 2^{n_1}.$$

For hash functions with more than two compression functions, we use the above algorithm to find fix-points (without B) of all but last compression functions, and then find the message block B. The time complexity in this case

is:

$$2 \cdot 2^{n_1} + 2^{n_1} + 2 \cdot n_2 \cdot 2^{n_2} + (n_2 + 1) \cdot 2^{n_2} + 2 \cdot n_3 \cdot n_2 \cdot 2^{n_3} + (n_3 \cdot n_2 + 1) \cdot 2^{n_3} + \ldots$$
$$+ \underbrace{2 \cdot \left(\prod_{i=2}^{k-1} n_i \right) \cdot 2^{n_{k-1}}}_{\text{a fix-point for } c_{k-1}(\cdot)} + \underbrace{\left[\left(\prod_{i=2}^{k-1} n_i \right) + 1 \right] \cdot 2^{n_k - 1}}_{\text{pre-image for } c_{k-1}(\cdot)} + \underbrace{\left[\left(\prod_{i=2}^{k} n_i \right) + 1 \right] \cdot 2^{n_k}}_{\text{pre-image for } c_k(\cdot)}$$

For comparison, we remind the reader that a naive approach to find a pre-image for such a hash function would be 2^n for $n = \sum_{i=1}^{k} n_i$.

4 Extending the Herding Attack

The extension of the herding attack is a non trivial operation. The herding attack stores several starting points, and then tries to reach one of them. This data-memory trade-off approach is very useful when the number of stored starting points is somewhat proportional to the number of possible chaining values. By storing 2^t starting points, the linking message requires 2^{n-t} trials (on average) to get connected to one of these values, where $n = \sum_{i=1}^{k} n_i$.

In the case of concatenated hash functions, if we want to store the same relative number of points, then one does not need to exploit the concatenation, however this would increase the memory complexity (and the pre-processing time) significantly. This fact becomes illustrated once the attacker has to find a linking message that fits all the intermediate values simultaneously.

Our algorithm is based on converging to the pre-computed paths of chaining values, one coordinate of the chaining values at a time. To perform the first convergences, we use a simple diamond structure as in the original herding attack. Then, the second coordinate is handled, by maintaining the knowledge of the first coordinate. This is done by using messages for the second diamond which are composed of multi-collisions in the first coordinate. Once the second coordinate converged to our known path, we can handle the next one, again using multi-collisions in the first two coordinates to maintain our knowledge of them.

There is another technical problem — while the first linking message to the first diamond construction is relatively unrestricted (besides its length), once we fix the first coordinate of the hash function, we cannot use another chaining message without taking into consideration the coordination that was fixed. Thus, it might seem that for the following compression functions we may need to store all possible starting points (of the compression function that we currently work with). This approach works, but is far from optimal.

We solve this problem by adding a series of linking blocks (again, based on multi-collisions), that ensures that independent of the chaining value in the second coordinate just at the end of the first diamond structure, we shall find one of the starting points to the second diamond. The process can then be repeated as many times as needed. Figure 4 depicts the structure for the case of two compression functions.

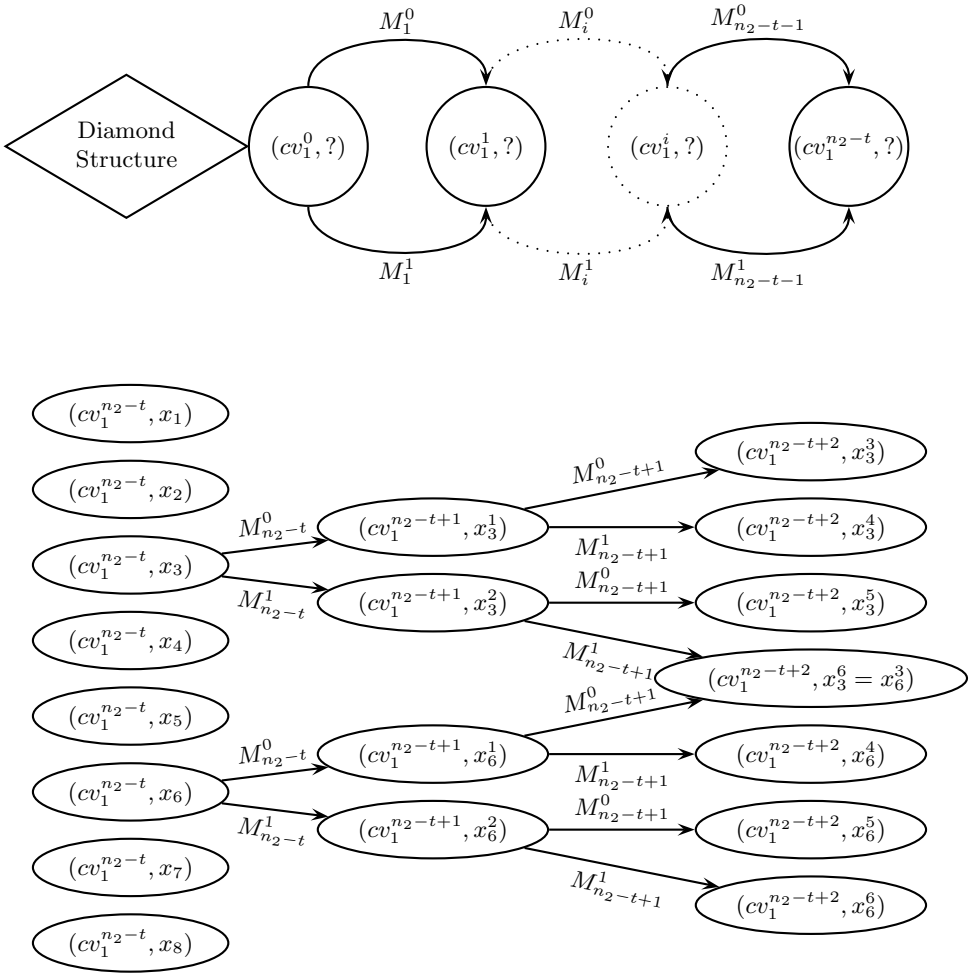

Fig. 2. The Main Idea Behind a Diamond Structure for a Hash Function of the Form $h(x) = h_1(x)||h_2(x)$.

Thus, the herding pre-processing algorithm (given for the case of two hash functions) is as follows:

1. **Constructing the first diamond:** Choose 2^t chaining values for $c_1(\cdot)$, and build from them the diamond structure of the herding attack. Denote the resulting chaining value by cv_1^0.
2. **Constructing a link between the diamonds:** For $i = 0, \ldots, (n_2 - t) - 1$ find two message blocks M_i^0 and M_i^1 such that $c_1(cv_1^i, M_i^0) = c_1(cv_1^i, M_i^1) = cv_1^{i+1}$.
3. **Constructing the second diamond:**

(a) Choose 2^t chaining values x_i for $c_2(\cdot)$.
(b) Generate $2^{n_2/2-t/2+1/2}$ multi-collisions in $c_1(\cdot)$. For $i = n_2 - t \ldots, (n_2 - t) + (n_2/2 - t/2 - 1/2)$ find two message blocks M_i^0 and M_i^1 such that $c_1(cv_1^i, M_i^0) = c_1(cv_1^i, M_i^1) = cv_1^{i+1}$.
(c) For each chaining value of $c_2(\cdot)$ try all possible sequences of messages $M^{e_{n_2-t}e_{n_2-t+1}\ldots e_{(3/2)(n_2-t)-1/2}} = M_{n_2-t}^{e_{n_2-t}}||M_{n_2-t+1}^{e_{n_2-t+1}}||\ldots||M_{(3/2)(n_2-t)-1/2}^{e_{(3/2)(n_2-t)-1/2}}$.
(d) From the generated 2^{t-1} chaining values for $c_2(\cdot)$, repeat Steps 2–3 with longer multi-collisions (to compensate for the smaller number of points) according to the diamond generation algorithm.

4. Denote the the resulting digest value by (cv_1, cv_2).
5. Concatenate a message block containing the right message padding for the Merkle-Damgård strengthening (if needed). Store the message block, and output the digest value as the target message value.

The online computational phase is as follows (again, given for the case of two hash functions):

1. Choose 2^{n_1-t} messages and try to obtain one of the 2^t starting points for $c_1(\cdot)$.
2. Once such a message is found, search over the 2^{n_2-t} combinations of $\{M_i^0, M_i^1\}$ for $i = 0, \ldots n_2 - t - 1$ of length $n_2 - t$, until one of the 2^t starting points stored in the second diamond structure is found.
3. Use the stored information in the diamond structure to obtain the full message.

The pre-processing phase requires $2^{n_1/2+t/2}$ for the generation of the first diamond structure. Then a set of 2^{n_2-t} multi-collisions are generated, in time complexity of $(n_2 - t) \cdot 2^{n_1/2}$. In the generation of the second diamond, we first generate $n_2/2 - t/2 + 1/2$ multi-collisions in $c_1(\cdot)$, in time complexity of $(n_2/2 - t/2 + 1/2) \cdot 2^{n_1/2}$. Then, the diamond algorithm is called again, but this time using messages of $n_2/2 - t/2 + 1/2$ blocks (rather than of one block). Thus, the time complexity of this step is $(n_2/2 - t/2) \cdot 2^{n_2/2+t/2}$.

During the process of generating the second diamond we slowly increase the length of the processed message block. This has almost no affect on the time complexity of the proper diamond structure generation, but it requires generating more multi-collisions. The total number of multi-collisions is

$$(n_2 - t)/2 + 1/2 + (n_2 - t)/2 + 3/2 + \ldots + n_2 = (n_2 + (n_2 - t)/2 + 1/2)/2 \cdot (t + 1)/2 \approx (3n_2 - t)/4 \cdot t/2 = 3/8n_2 \cdot t - t^2/8.$$

Thus, the total number of calls to the compression functions in the pre-processing phase is $2^{n_1/2+t/2} + t/8 \cdot (3n_2 - t) \cdot 2^{n_1/2} + (n_2/2 - t/2) \cdot 2^{n_2+t/2} \approx 2^{n_1/2+t/2} + (n_2/2 - t/2) \cdot 2^{n_2/2+t/2}$.

It is sufficient to store in each layer of the second diamond structure the index of the message sequence that leads to the collision between the $c_2(\cdot)$ chaining values. Thus, the memory complexity of the second diamond structure is similar to the memory required to store such a diamond structure for a hash function

of n_2 output bits (up to some small memory required for storing M_i^0 and M_i^1 for the respective i values) .

The online phase requires 2^{n_1-t} calls for $c_1(\cdot)$ (we note that computing c_2 is not necessary before we find a message that achieves one of the required starting points). Then, the attacker has to try 2^{n_2-t} calls to $c_2(\cdot)$ with messages of length n_2-t. Thus, the total time complexity of the online phase is $2^{n_1-t}+(n_2-t)\cdot 2^{n_2-t}$ compression function calls.

For comparison, applying the basic herding attack without using the concatenated nature, the attack would require $2^{2s} \cdot 2^{(n_1+n_2)}/2$ calls for the joint compression function in the pre-processing phase (to generate 2^s starting points), and $2^{(n_1+n_2)-s}$ calls to find the linking message.

For multiple number of hash functions, one has to use in the pre-processing phase the multi-collision approach that was used in Section 3.3. The corresponding time complexity of the attack for k compression functions in that case is roughly

$$2^{n_1/2+t/2} + (n_2/2 - t/2) \cdot 2^{n_2/2+t/2} + \ldots + \left(\prod_{i=2}^{k-1} n_i/2\right) \cdot (n_k - t/2) \cdot 2^{n_k/2+t/2}$$

The online computational phase is only slightly altered, and has the total time complexity of $2^{n_1-t} + (n_2 - t) \cdot 2^{n_2-t} + (n_3 - t) \cdot n_2 \cdot 2^{n_3-t} + \ldots + (n_k - t) \cdot \left(\prod_{i=2}^{k-1} n_i\right) \cdot 2^{n_k-t}$ compression function calls.

5 Summary

In this work we have shown that the herding attack can be extended to cases where there is dependence between concatenated compression functions, as long as the flow of dependence is in one direction. Following our findings along with Joux's initial observations we recommend hash functions designers to have good diffusion in case they prefer to use several data paths (to reduce implementation costs).

A problem that is still open is the extension of the long second pre-image attack to such hash functions. Another open problem is how to deal with the case where two data paths affect each other, as suggested by many recent proposals.

References

1. Eli Biham, Rafi Chen, *Near-Collisions of SHA-0*, Advances in Cryptology, proceedings of CRYPTO 2004, Lecture Notes in Computer Science 3152, pp. 290–305, Springer-Verlag, 2004.
2. Eli Biham, Rafi Chen, Antoine Joux, Patrick Carribault, Christophe Lemuet, William Jalby, *Collisions of SHA-0 and Reduced SHA-1*, Advances in Cryptology, proceedings of EUROCRYPT 2005, Lecture Notes in Computer Science 3621, pp. 36–57, 2005.

3. Florent Chabaud, Antoine Joux, *Differential Collisions in SHA-0*, Advances in Cryptology, proceedings of CRYPTO 1998, Lecture Notes in Computer Science 1462, pp. 56–71, Springer-Verlag, 1998.
4. Don Coppersmith, *Another Birthday Attack*, Advances in Cryptology, proceedings of CRYPTO 1985, Lecture Notes in Computer Science 218, pp. 14–17, Springer-Verlag, 1986.
5. Ivan Damgård, *A Design Principle for Hash Functions*, Advances in Cryptology, proceedings of CRYPTO 1989, Lecture Notes in Computer Science 435, pp. 416–427, Springer-Verlag, 1990.
6. Richared D. Dean, *Formal Aspects of Mobile Code Security.*, Ph.D. dissertation, Princeton University, 1999.
7. Praveen Gauravaram, William Millan, Ed Dawson, Matt Henricksen, Jaunma Gonzáles Nieto, Kapali Viswanathan, *Constructing Secure Hash Functions by Enhancing Merkle-Damgård Construction (Full Version)*, available online at `http://www.isi.qut.edu.au/research/publications/technical/qut-isi-tr-2006-013.pdf`.
8. Jonathan J. Hoch, Adi Shamir, *Breaking the ICE — Finding Multicollisions in Iterated Concatenated and Expanded (ICE) Hash Functions*, preproceedings of Fast Software Encryption 2006, pp. 199–214, 2006.
9. Antoine Joux, *Multicollisions in Iterated Hash Functions*, Advances in Cryptology, proceedings of CRYPTO 2004, Lecture Notes in Computer Science 3152, pp. 306–316, Springer-Verlag, 2004.
10. John Kelsey, Tadayoshi Kohno, *Herding Hash Functions and the Nostradamus Attack*, preproceedings of Cryptographic Hash Workshop, held in NIST, Gaithersburg, Maryland, 2005.
11. John Kelsey, Bruce Schneier, *Second Preimages on n-Bit Hash Functions for Much Less than* 2^n, Advances in Cryptology, proceedings of EUROCRYPT 2005, Lecture Notes in Computer Science 3494, pp. 474–490, Springer-Verlag, 2005.
12. Ralph C. Merkle, *Secrecy, Authentication, and Public Key Systems*, UMI Research press, 1982.
13. Ralph C. Merkle, *One Way Hash Functions and DES*, Advances in Cryptology, proceedings of CRYPTO 1989, Lecture Notes in Computer Science 435, pp. 428–446, Springer-Verlag, 1990.
14. US National Bureau of Standards, *Secure Hash Standard*, Federal Information Processing Standards Publications No. 180-2, 2002.
15. Xiaoyun Wang, Xuejia Lai, Dengguo Feng, Hui Chen, Xiuyuan Yu, *Cryptanalysis of the Hash Functions MD4 and RIPEMD*, Advances in Cryptology, proceedings of EUROCRYPT 2005, Lecture Notes in Computer Science 3494, pp. 1–18, 2005.
16. Xiaoyun Wang, Yiqun Lisa Yin, Hongbo Yu, *Finding Collisions in the Full SHA-1*, Advances in Cryptology, proceedings of CRYPTO 2005, Lecture Notes in Computer Science 3621, pp. 17–36, 2005.
17. Xiaoyun Wang, Hongbo Yu, *How to Break MD5 and Other Hash Functions*, Advances in Cryptology, proceedings of EUROCRYPT 2005, Lecture Notes in Computer Science 3494, pp. 19–35, 2005.
18. Xiaoyun Wang, Hongbo Yu, Yiqun Lisa Yin, *Efficient Collision Search Attacks on SHA-0*, Advances in Cryptology, proceedings of CRYPTO 2005, Lecture Notes in Computer Science 3621, pp. 1–16, 2005.